JOURNAL OF COMPUTER SCIENCES INSTITUTE

JCSI 15 (2020) 184-188
Received: 30 March 2020
Accepted: 6 April 2020

Containerization methods in internet service applications

Metody konteneryzacji w zastosowaniach usług internetowych

Krzysztof Ferenc*

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

This work describes the most important methods of isolating running processes of selected software on one physical machine without using full virtualization of the operating system. It includes a comparison of the basic components of each of the methods described, including the features and interface, as well as the results of performance testing in a test environment prepared for this purpose. Research is based on running an identical application and comparing the use of basic system resources, as well as the time of launching containers. It may also be important to compare the methods studied to classical virtualization. At the end of the work, conclusions are described that may be relevant when choosing a method in future projects depending on requirements and needs.

Keywords: containers; virtualization; isolation; docker

Streszczenie

Niniejsza praca opisuje najważniejsze metody izolacji uruchomionych procesów wybranego oprogramowania na jednej maszynie fizycznej bez użycia do tego celu pełnej wirtualizacji systemu operacyjnego. Zawiera porównanie podstawowych składników każdej z opisanych metod, w tym cechy i interfejsu, a także wyniki badania wydajności w przygotowanym do tego celu środowisku testowym. Badania oparte są na uruchamianiu identycznej aplikacji i porównywaniu wykorzystania podstawowych zasobów systemowych, a także czasu uruchamiania kontenerów. Istotne jest tu również porównanie badanych metod do klasycznej wirtualizacji. Na końcu pracy opisane są wnioski, które mogą być istotne przy wyborze metody w przyszłych projektach w zależności od wymagań i potrzeb.

Słowa kluczowe: kontenery; wirtualizacja; izolacja; docker

*Corresponding author

Email address: **krzysztof.ferenc1@pollub.edu.pl** (K. Ferenc)

1. Wstęp

Przy tworzeniu nowoczesnych serwisów internetowych i aplikacji użytkowych dostępnych w sieci coraz większą rolę odgrywają mikroserwisy jako sposób na modularyzację aplikacji działających w środowisku produkcyjnym. Mikroserwisy jako wzorzec projektowy [1] używany przez architektów oprogramowania opiera się na koncepcie kontenerów, a te z kolei na wirtualizacji.

Jedną z najważniejszych zalet mikroserwisów jest ich uniwersalność, dzięki czemu idealnie nadają się do odtwarzania jednolitego środowiska niezależnie od fizycznej maszyny, na której zostają uruchomione. Taka konstrukcja sprawia, że nierzadko wykorzystywane są również przez programistów i testerów podczas wytwarzania oprogramowania.

Niniejsza praca skupia się na omówieniu wybranych sposobów konteneryzacji aplikacji i usług internetowych, w których mogą być one dzielone na mikroserwisy i separowane od siebie na poziomie systemu operacyjnego.

Poza samym omówieniem cech charakterystycznych dla konteneryzacji w odróżnieniu od innych podobnych rozwiązań, praca zawiera również porównanie wybranych dostępnych metod konteneryzacji, a także testy wydajnościowe dla przykładowego przypadku testowego.

Postawiono następującą tezę badawczą:

Kontenery w porównaniu z klasycznym rozwiązaniem wirtualizacji całego systemu operacyjnego wykazują się znacznie większą wydajnością przy uruchamianiu usług internetowych. Wykorzystanie procesora oraz pamięci operacyjnej przez maszynę wirtualną jest wyższe, a czas uruchamiania dłuższy.

2. Przegląd dostępnych rozwiązań

Według danych firmy Sysdig, Inc. [2] najpopularniejszym konteneryzatorem używanym w środowiskach produkcyjnym w 2018 roku był Docker z udziałem na poziomie 83% w porównaniu z konkurencyjnymi rozwiązaniami [3]. Ten wynik wskazuje na znaczny spadek z poziomu 99% w 2017 według danych tej samej firmy. Te same dane określają także procentowy udział w 2018 roku narzędzi takich jak: CoreOS rkt – 12%, Apache Mesos – 4% i LXC – 1%.

Do porównania spośród wymienionych najpopularniejszych narzędzi [3] wybrane zostały Docker i rkt z uwagi na najwyższą popularność, a także kontenery LXC/LXD, które są najstarszym i najdłużej używanym z wymienionych rozwiązań. Oprogramowanie Apache Mesos nie zostało opisane w niniejszej pracy z powodu relatywnie krótkiego istnienia (od lipca 2016 roku), a także głównego nastawienia na zastosowanie w dużych centrach danych i zarządzania klastrami maszyn [4], co stanowi kontrast do pozostałych systemów, które

uwzględniają przypadki użycia w ramach jednej maszyny fizycznej.

3. Metodyka badawcza

3.1. Stanowisko badawcze

Każde z porównywanych rozwiązań posiada odmienne wymagania w zakresie warstwy sprzętowej, jak i systemu operacyjnego, na którym narzędzie konteneryzacji jest wpierane. To co je łączy w zakresie wymagań to fakt, że działają natywnie na systemach bazujących na Linuxie, który to jest najpopularniejszy w środowiskach serwerowych [5]. Z uwagi na to, do celów badania obsługi i wydajności poszczególnych rozwiązań wybrany został system Arch Linux jako system operacyjny hosta.

Głównym powodem tej decyzji jest prostota przenoszenia oprogramowania z innych systemów bazujących na jądrze Linux, a także możliwość stworzenia lekkiej instalacji z minimalną liczbą dodatkowego oprogramowania, które poprzez nierównomierne zużycie zasobów mogłoby zaburzyć odczyty wydajności systemów konteneryzacji.

Do testów została wykorzystana fizyczna maszyna w formie komputera stacjonarnego o architekturze procesora x86_64, nazywanej często amd64. Jest to jedyny możliwy wybór, który jest wspierany przez wszystkie konteneryzatory.

3.2. Narzędzia badawcze

Pozyskiwanie wyników ma charakter eksperymentalny, poprzez uruchomienie przykładowej aplikacji internetowej w formie obrazu obsługiwanego przez systemy konteneryzacji jako działający kontener. Następnie przy użyciu programów pomiarowych zostanie zmierzony czas i zasoby używane przez każdy z kontenerów.

Do celów uruchamiania aplikacji internetowej w kontenerze posłuży prosty serwer działający na frameworku Node.js, który odpytany na odpowiednim adresie i porcie zwraca odpowiedź tekstową. Kod programu przedstawia Listing 1.

Listing 1: Przykład kodu uruchamiającego serwer aplikacji internetowej

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at
http://${hostname}:${port}/`);
});
```

Wybór wspomnianego frameworka opiera się o wysoką dostępność i łatwość użycia w dowolnym środowisku. Większość dokumentacji i poradników wybiera również ten konkretny przykład do pokazania łatwości w użyciu danego rozwiązania wirtualizacji.

Do pomiaru czasu wykonywania polecenia uruchamiania kontenerów wykorzystany zostanie wbudowany w większość systemów bazujących na Linuxie program GNU time [6].

Pomiar wykorzystanych zasobów procesora i pamięci systemowej wykonany zostanie natomiast z użyciem wbudowanych narzędzi diagnostycznych w każdym z systemów konteneryzacji lub z użyciem wbudowanego w system operacyjny programu ps, którym można monitorować zarówno wykorzystanie przez proces procesora maszyny gospodarza, jak i użycie pamięci.

Właściwości badania statystycznego dobrane zostaną w taki sposób, aby jak najmniej wpłynąć na wykorzystanie zasobów przez narzędzia pomiarowe, a jednocześnie możliwe było wskazanie charakterystyki zmian badanych wartości. Na końcu uzyskane wyniki zostaną uśrednione dla wszystkich wykonanych uruchomień danego kontenera i czasu, następnie porównane razem z wynikami uzyskanymi dla klasycznej metody wirtualizacji jako próba porównawcza.

4. Porównanie funkcji kontenerów

Różnice między opisywanymi systemami konteneryzacji pojawiają się na warstwie obsługiwanych metod izolacji zasobów, a także samej architektury narzędzi. Tabela 1 zawiera zestawienie wspieranych funkcji w omawianych narzędziach.

Tabela 1: Porównanie funkcji systemów konteneryzacji

Funkcja	LXC/LXD	Docker	rkt
Centralna usługa zarządzająca kontenerami	Tak	Tak	Nie
Izolacja systemu plików	Tak	Tak	Tak
Ograniczanie przestrzeni dyskowej	Nie	Nie	Nie
Ograniczanie przepustowości przestrzeni dyskowej	Nie	Tak	Nie
Ograniczanie pamięci systemowej	Tak	Tak	Tak
Ograniczanie mocy obliczeniowej	Tak	Tak	Nie
Izolacja sieci	Tak	Tak	Tak
Wirtualizacja zagnieżdżona	Tak	Tak	Nie
Migracja na żywo	Tak	Nie	Nie
Izolacja uprawnień systemu plików	Tak	Tak	Tak

5. Interfejs

Istnieją dwa główne rodzaje interfejsów używanych do komunikacji z usługą zarządzania kontenerami – tekstowy i graficzny.

Interfejs tekstowy jest uniwersalny dla każdego z rozwiązań, natywnie wspierany, a także jest podstawą przy rozpoczynaniu nauki z mikroserwisami. Zaletą jego używania jest przejrzystość prezentacji danych i precyzja podczas wykonywania operacji. Początkujących administratorów lub też ludzi nieobeznanych z terminalem może odrzucić pozorna archaiczność tego sposobu interakcji, ponieważ nie jest ona najbardziej

intuicyjna i wymaga wcześniejszego przygotowania w postaci przeczytania fragmentu dokumentacji.

Z przedstawionych metod obsługi tworzenia obrazu wynika, że czas i ilość kroków znacznie różni się w zależności od wybranego systemu konteneryzacji. Dla Dockera wystarczy jeden plik konfiguracyjny i jedno polecenie do budowania obrazu. System rkt udostępnia narzędzie do tworzenia pliku konfiguracyjnego bez potrzeby tworzenia go ręcznie i nie wymaga dodatkowego polecenia budowania obrazu. LXD posiada najbardziej skomplikowaną procedurę wymagającą ręcznego pobrania, stworzenia systemu plików i spakowania go do archiwum, następnie stworzenia i spakowania pliku konfiguracyjnego, aby na końcu wywołać polecenie tworzenia obrazu.

Niezmiennie prosta jest natomiast obsługa już utworzonego obrazu. Przestawione tutaj przykładowo polecenia uruchamiania kontenerów analogicznie funkcjonują przy wykonywaniu innych operacji, jak włączanie i wyłączanie kontenera, usuwanie kontenerów i obrazów, listowanie i uruchamianie dodatkowych procesów w działających kontenerach.

Graficzne zarządzanie jest zdecydowanie rzadziej stosowaną formą interakcji i często nie oferuje pełni możliwości w porównaniu do klasycznego terminala tekstowego. Z pośród badanych metod konteneryzacji jedynie Docker posiada szeroki wybór narzędzi, które pozwalają na większość podstawowych operacji zarządzania obrazami, sieciami, wolumenami i samymi kontenerami w tym ich tworzeniem, monitorowaniem i usuwaniem.

6. Wydajność

W efekcie dobrania odpowiednich właściwości badania statystycznego uzyskane dane opierają się na dwudziestu próbach uruchomienia każdego z kontenerów w nieobciążonym środowisku testowym na czas wystarczający do pełnej gotowości izolowanej usługi. Dane o zużywanych zasobach zbierane były co sześć sekund, dzięki czemu wpływ narzędzi pomiarowych na działanie kontenerów był marginalny.

Najdłuższy czas uruchomienia usługi do stanu pełnej gotowości wyniósł 125 sekund, co wystąpiło na maszynie wirtualnej stanowiącej próbę porównawczą. Z tego powodu czas wszystkich badań został ustalony na najbliższą wartość pomiarową przekraczającą ten czas, czyli 126 sekund, na co przypadają 22 pomiary zużywanych zasobów.

Po wyznaczonym czasie każdy z kontenerów zostawał wyłączony, a następnie włączony kolejny.

6.1. Czas uruchamiania

Z wykresu na Tabeli 2 można odczytać bardzo podobne wyniki czasów uruchamiana kontenerów przy zastosowania do tego celu narzędzi Docker i LXD. Ponadto wahania między najkrótszym, a najdłuższym czasem uruchomienia przy próbie nie przekraczały odpowiednio 0,29 i 0,27 sekundy.

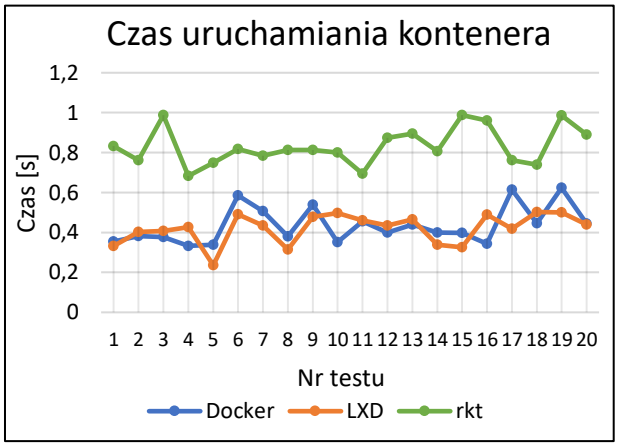
Nieco inaczej sytuacja wygląda w przypadku rkt i tutaj czasy uruchamiania są wyraźnie większe, co przekłada się na powolniejszy start usług. Nie jest to natomiast ogromna różnica. Ekstrema również są nieco bardziej oddalone – o dokładnie 0,3 sekundy.

Tabela 2: Wyniki badania czasu uruchamiania [s]

Nr. testu	Docker	LXD	rkt	KVM
1	0,35	0,33	0,83	74,47
2	0,38	0,40	0,76	77,89
3	0,38	0,41	0,99	70,12
4	0,33	0,43	0,68	74,42
5	0,34	0,24	0,75	74,63
6	0,59	0,49	0,82	79,36
7	0,51	0,43	0,78	81,20
8	0,38	0,31	0,81	80,29
9	0,54	0,48	0,81	77,38
10	0,35	0,50	0,80	79,61
11	0,46	0,46	0,69	76,55
12	0,40	0,44	0,87	77,15
13	0,44	0,46	0,89	75,73
14	0,40	0,34	0,81	78,27
15	0,40	0,33	0,99	70,13
16	0,34	0,49	0,96	77,49
17	0,61	0,42	0,76	75,93
18	0,45	0,50	0,74	79,59
19	0,62	0,50	0,99	77,19
20	0,44	0,44	0,89	72,21

Testy dla maszyny wirtualnej (KVM) wykazały ogromną różnicę między czasem uruchamiania. W porównaniu do najdłuższego czasu uruchamiania kontenera - 0,99 sekundy, najkrótszy czas uruchamiania maszyny wirtualnej z identyczną aplikacją wyniósł 70,12 sekundy, co stanowi niemal 70-krotną różnicę (dokładnie 6982,83%).

Wyniki dla systemów konteneryzacji przedstawione zostały na rysunku 1.



Rysunek 1: Czas uruchamiania pojedynczego kontenera - porównanie

6.2. Wykorzystanie mocy procesora

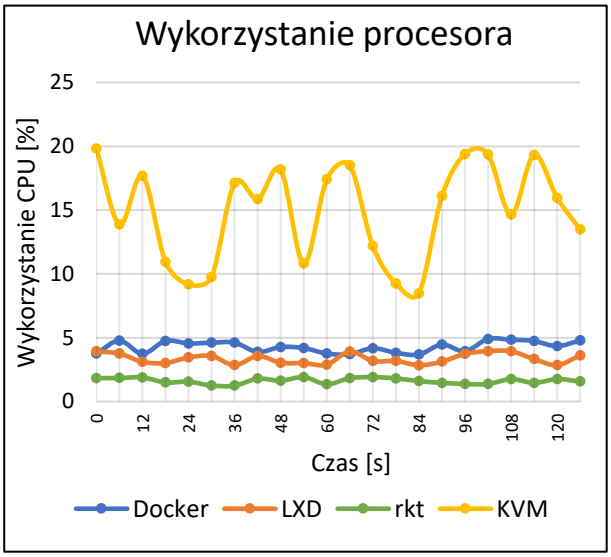
Wykorzystywana moc procesora (Tabela 3), reprezentowana w procentach, wskazuje na podobne, niewielkie wahania dla każdego z systemów konteneryzacji – od 0,67 % dla rkt, do 1,22% dla Dockera. Inaczej jest jednak w przypadku maszyny wirtualnej, gdzie zauważalne są różnice między najniższym i najwyższym wykorzystaniem procesora na poziomie 11,35%.

Faktem jest także to, że narzędzie rkt najmniej obciąża procesor, co może być wynikiem braku centralnej usługi zarządzania kontenerami.

Tabela 3: Wyniki badania wykorzystania mocy procesora [%]

Czas [s]	Docker	LXD	rkt	KVM
0	3,75	3,92	1,83	19,81
6	4,77	3,76	1,84	13,87
12	3,74	3,11	1,89	17,69
18	4,75	3,02	1,49	10,94
24	4,55	3,45	1,55	9,17
30	4,61	3,57	1,25	9,76
36	4,61	2,85	1,24	17,13
42	3,88	3,57	1,79	15,85
48	4,27	3,04	1,63	18,18
54	4,19	3,00	1,91	10,80
60	3,76	2,88	1,35	17,41
66	3,70	3,92	1,83	18,52
72	4,17	3,17	1,91	12,18
78	3,81	3,18	1,81	9,25
84	3,68	2,84	1,59	8,47
90	4,46	3,13	1,46	16,10
96	3,94	3,73	1,37	19,39
102	4,90	3,95	1,37	19,37
108	4,85	3,93	1,75	14,64
114	4,74	3,32	1,46	19,32
120	4,34	2,85	1,76	15,95
126	4,79	3,61	1,56	13,48

Obserwacje te można zauważyć również na wykresie z Rysunku 2. Widoczne są wyraźne wzrosty i spadki funkcji powiązanej z wynikami maszyny wirtualnej (KVM).



Rysunek 2: Wykorzystanie mocy procesora przez pojedynczy kontener - porównanie

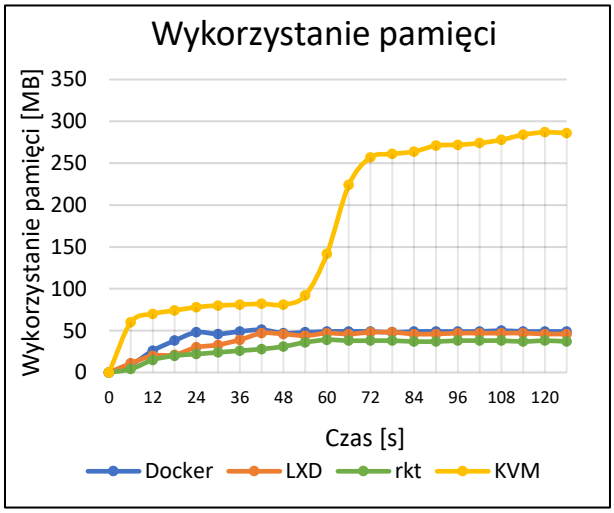
6.3. Wykorzystanie pamięci operacyjnej

W tabeli 4 w każdym przypadku widać wzrost wykorzystania mocy procesora w czasie działania instancji kontenera. Dla systemów konteneryzacji wykorzystanie to stabilizuje się po około 54-ech sekundach od uruchomienia, po którym to czasie nie zwiększa się. Maszyna wirtualna natomiast poza znacznie większym wykorzystaniem pamięci operacyjnej charakteryzuje się dwoma skokami w zapotrzebowaniu – w 6-tej i 60-tej sekundzie od uruchomienia.

Tabela 4: Wyniki badania wykorzystania pamięci [MB]

Czas [s]	Docker	LXD	rkt	KVM
0	0	0	0	0
6	9	11	4	60
12	26	20	15	70
18	38	21	20	74
24	48	30	22	78
30	46	33	24	80
36	49	39	26	81
42	51	47	28	82
48	47	46	31	81
54	48	44	36	92
60	49	47	39	142
66	49	46	38	224
72	49	48	38	257
78	48	48	38	261
84	49	46	37	264
90	49	46	37	271
96	49	47	38	272
102	49	47	38	274
108	50	47	38	278
114	49	47	37	284
120	49	46	38	287
126	49	46	37	286

Wizualna reprezentacja danych na Rysunku 3 dodatkowo uwidacznia skoki w wykorzystaniu zasobów pamięci dla maszyny wirtualnej, a także szybką stabilizację dla konteneryzatorów.



Rysunek 3: Wykorzystanie pamięci operacyjnej – porównanie

7. Wnioski

Przeprowadzone badania jasno wskazują, że istnieją różnice między badanymi rozwiązaniami z zakresu konteneryzacji aplikacji internetowych. Dla uruchomionej aplikacji w różnych systemach konteneryzacji wykorzystywane były różne ilości dostępnych zasobów systemowych maszyny gospodarza.

Wyniki wykorzystania mocy procesora i pamięci operacyjnej pokazują, że największe zapotrzebowanie z badanych rozwiązań ma konteneryzator Docker. Pomimo jednak tego, czas uruchamiania jest znacznie krótszy od rkt, czyli systemu o najmniejszych wymaganiach sprzętowych.

Najbardziej zbalansowanym pod względem używanych zasobów, jak i czasu uruchamiania jest system konteneryzacji LXD, którego zapotrzebowanie na moc obliczeniową procesora jest o 70% wyższe od rkt, jednak o 24% niższe od Dockera. Podobnie wygląda sytuacja przy zapotrzebowaniu na pamięć operacyjną, gdzie LXD jest o 26% mniej wydajny od rkt, jednak o 11% wydajniejszy od Dockera

Cechą wyróżniającą kontenery rkt jest ich długi czas uruchamiania w porównaniu do innych rozwiązań. Oznacza to, że kontenery te nie nadają się najlepiej do krótko żyjących procesów. Z uwagi jednak, że rkt wykazuje najniższe zużycie zasobów – świetnie nada się w roli konteneryzatora do długożyjących procesów, co w kontekście aplikacji internetowych może być wykorzystane do uruchomienia API, które musi być stale dostępne do użycia.

Kwestią bardziej indywidualną może być interfejs, który udostępnia każdy z systemów izolacji aplikacji. Komunikacja z oprogramowaniem wygląda podobnie w każdym przypadku z uwagi na ustandaryzowany sposób wykorzystania wiersza poleceń. Udogodnieniem mogą być także polecenia podzielone na kategorie, które jednak w pełnym zakresie dostępne są jedynie w narzędziu Docker, jednak częściowo występują w każdym z badanych rozwiązań.

Istotną kwestią, jaką należy jednak brać pod uwagę jest popularność. W przypadku konteneryzacji zdecydowaną przewagę osiągnął Docker, który w ciągu ostatnich lat stał się liderem, a przez wielu również jedynym znanym sposobem izolacji aplikacji bez używania do tego celu zasobożernych maszyn wirtualnych.

Ponieważ popularność często niesie za sobą również wsparcie przez producentów oprogramowania w postaci gotowych do użycia obrazów, może okazać się, że Docker jest najprostszym sposobem uruchomienia istniejącej na rynku aplikacji internetowej. Mniejsze znaczenie ma to w przypadku tworzenia własnego obrazu dla kontenera.

Bez względu na wybrane narzędzie konteneryzacji, wyniki badań udowadniają, że charakteryzują się one większą wydajnością w porównaniu do klasycznego rozwiązania izolacji aplikacji, jakim jest pełna wirtualizacja systemu operacyjnego, co potwierdza postawioną we wstępie tezę badawczą.

Istnieją podobne badania [7] wydajności systemów konteneryzacji w porównaniu z maszynami wirtualnymi. Z uwagi na różnice w zastosowanych scenariuszach nie jest możliwe jednoznaczne porównanie wyników, jednak wnioski są podobne i wskazują, że użycie kontenera do udostępnienia usługi jest bardziej wydajne niż wykorzystanie maszyny wirtualnej, co również udowadnia postawioną tezę.

Literatura

[1] S. Newman, Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 2015.

[2] E. Carter, 2018 Docker usage report, https://sysdig.com/blog/2018-docker-usage-report/ [24.03.2020].

[3] B. Doerrfeld, 5 Container Alternatives to Docker, https://containerjournal.com/topics/container-ecosystems/5-container-alternatives-to-docker/ [24.03.2020].

[4] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, I. Stoica, Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center, University of California, Berkeley, 2010, https://people.eecs.berkeley.edu/~alig/papers/mesos.pdf [24.03.2020].

[5] W3Cook.com, OS Market Share and Usage Trends : https://web.archive.org/web/20150806093859/http://www.w3cook.com/os/summary/ [24.03.2020].

[6] GNU Time – GNU Project – Free Software Foundation, https://www.gnu.org/software/time/ [29.01.2020].

[7] B. Russell, KVM and docker LXC Benchmarking with OpenStack Trends, https://www.slideshare.net/BodenRussell/kvm-and-docker-lxc-benchmarking-with-openstack?qid=5d451a56-0bbb-4c30-a411-8a20c83d4992 [24.03.2020].